並列プログラミング教育のための自学用Webアプリケーションスイートの試作

著者	若谷 彰良, 前田 利之
雑誌名	甲南大学紀要. 知能情報学編
巻	13
号	2
ページ	87-102
発行年	2021-02-10
URL	http://doi.org/10.14990/00003693

論　文

並列プログラミング教育のための自学用 Web アプリケーションスイートの試作

若谷彰良 [a], 前田利之 [b]

[a] 甲南大学 知能情報学部
神戸市東灘区岡本８－９－１, 658-8501
[b] 阪南大学 経営情報学部
大阪府松原市天美東５－４－３３, 580-8502

(受理日 2020 年 11 月 16 日)

概 要

現代のコンピュータシステムにおいて, 並列プログラミングは高速化のために必要不可欠な技術となっている. しかし, 逐次実行とは異なる考え方をプログラミングの中に含める必要があるので, 初学者にとっては, 学ぶ上での困難さを伴うものである. そこで, 本稿では, 並列プログラミングの主要技術である, OpenMP, POSIX スレッド, AVX, MPI, CUDA, OpenACC のプログラミング教育のための学習教材を, Web アプリケーションスイートとして試作するためのポリシーと, 完成した試作版の現状及びその予備評価について述べる. 本 Web アプリケーションスイートは, 初学者の自習用教材として用いるだけでなく, 反転授業における事前学習にも用いることができ, さらには, 新感染症対策の対面授業自粛期間における, オンライン講義の教材としても利用することができる. また, 予備評価の結果より, プログラミング演習の Web ツールは理解度の向上に寄与することが確認できる.

キーワード： PHP, 自動生成, 自動採点, 電子教材, OpenACC, OpenMP, MPI, AVX, CUDA, POSIX スレッド, COVID-19

1　はじめに

2020 年 1 月 16 日に, 日本で最初の新型コロナウィルス (COVID-19) の感染を確認して以来, COVID-19 に対する対策は, 日本だけでなく, 全世界的に緊急課題となり, 経済活動を含め, 様々なシチュエーションでその影響は大きくなってきている. 教育においては, 登校の自粛にともない, オンデマンド資料やビデオ会議システムを用いたオンライン講義が一般的になってきている. オンデマンド資料は, このような感染症対策だけでなく, 新しい教育スタイルのひとつとして注目されている反転授業 (flip teaching) [1], [2] において, 自宅での事前学習にも用いられる. また, 学習意欲のある学生が, より知識を高めるためにも, オンデマンド資料は有用なツールとなりうる. しかし, このオンデマンド型の学習形態を, プログラミング学習に用いる場合, 受動的な資料の視聴だけでは学習内容は身に付きにくく,

プログラミング体験を用いた, 能動的な学習ツールが必要不可欠である. よって, プログラミング演習を体験できるツールは重要である.

一方, プロセッサを構成する半導体の微細化が進み, 10 nm から 7 nm のプロセスルールで設計がされつつある. これに伴い, LSI 上に実装されるトランジスタの集積度が高まり, マイクロプロセッサやグラフィックプロセッサ (GPU) には多くのプロセシングコア (コア) が搭載されるようになり, 並列プログラミングは重要な技術になっている. 特に GPU 上には 5000 個を超えるコアが搭載され [3], ディープラーニングや数値計算など, グラフィックス以外のアプリケーション分野での利用も進んでいるが, そこで利用されるアプリケーションのプログラミングの抽象化が進んでいる. 例えば, ディープラーニングでは Python 言語で実装されたフレームワークを用いることが多いが, その代表の一つである Chainer [4] においては, 学習データを用いてモデルのパラメータの学習を行う過程の記述自体を “training.StandardUpdater” というメソッドを呼ぶだけで実現できる. ディープラーニングにおいては, 特に大量の学習データのフォワード処理をし, その後にバックワード処理でネットワークの重みの学習を繰り返すので, 大規模な計算量となり, GPU を前提とした実装が必要である. そこで, Chainer においては, 大量の学習データを複数の GPU に振り分けて並列処理によって性能を向上させるデータ並列を “training.ParallelUpdater” メソッドを呼ぶだけで実現できるようになっている. 抽象化のメリットは当然プログラムの簡素化があり, 一般的に困難であるとされる GPU プログラミングの詳細を, 利用者が学ぶ必要がないことがあげられるが, 利用者にとっては得られた性能が十分なのか否かが判断しにくく, さらなる性能向上のための最適化が可能なのかどうかが全く判断できないことになる. 抽象化は, 本来は, 中身を見えなくすることが目的ではなく, プログラミングの簡素化のためであると考える. したがって, 抽象化が進んだ現代においても, 初学者が, そのレベルあった並列化プログラミングの理解は, プログラマとしては必須な修得技術の一つと考えられる.

プログラミング教育を, 自宅学習と組み合わせた試みはあり, Maher らは, Web アプリケーション開発の教育をオンラインビデオとクラスでのグループワークを組み合わせ, その評価を行っている [5]. また, Radošević らは, プログラミング教育におけるオンライン教材の自動生成について研究している [6]. 彼らの手法は部分的には成功しているが, 運用する教員に高いスキルを要求し, 採点の手法に問題がある. また, いくつかの先行研究では課題の自動採点については取り組まれているが, 課題自体の自動生成には触れられていない [7], [8].

本稿では, 必要性が急激に増加している並列プログラミングに対して, 複数の並列プログラミング環境の学習を一括して行える, 自学用 Web アプリケーションスイートの試作の現状 [9] と予備評価について述べる. これにより, 対面講義自粛期間中の自宅学習や, 反転講義の事前学習及びプログラミングの自習教材として利用することができる. 本アプリケーションスイートには, 並列プログラミングの代表として, OpenMP, POSIX スレッド, AVX, MPI, CUDA, OpenACC のプログラミング教育のための学習教材で構成され, 能動的なプログラミング体験が可能な Web ツールが含まれている. 以降の章において, 本アプリケーションスイートが初学者にとって有用なツールとなるための作成ポリシーと, いくつかの並列プログラミングに対する予備評価について, MPI と OpenACC を中心に述べる.

2 並列プログラミング

2.1 概要

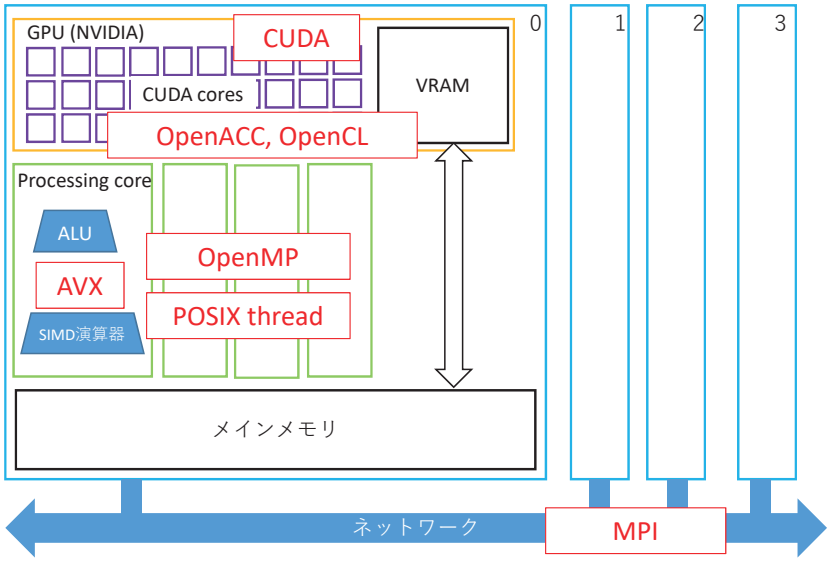


図 1: 並列コンピュータシステム

現在のコンピュータは, 図 1 に示すように, その要素に様々な並列性を含む構成となっている. まず, プロセッサは複数のプロセシングコアからできており, そのプロセシングコアの中には, SIMD 演算器があり, 一つの命令で複数の演算が同時に実行できる. さらに, プロセシングコアの中には, ALU が複数あり, 命令スケジューラにより, 同時実行可能な命令が, 同時に実行されている. さらに, GPU が備えられおり, その中には数百から数千のコアが実装されて, 専用のメモリである VRAM を用いて, グラフィック処理だけでなく, 汎用計算にも利用されている. このようなコンピュータをネットワークで接続して, クラスタコンピュータとして, さらに性能向上を目指すようなシステムもあり, 昨今のスーパーコンピュータは, このような構成で構築されている.

このようなコンピュータシステムにおいて, 複数 ALU を命令スケジューラを用いて同時並行実行する部分は, ハードウェアが命令列の解析をすることで実現しているが, それ以外の並列性は, ソフトウェアによる実装により実現されている.

プロセシングコア内の SIMD 演算器の利用は, コンパイラが自動的に利用する場合もあるが, 独自の API を用いてプログラミングする手法があり, Intel のプロセッサでは AVX [10] というライブラリが利用されている. これは, プロセッサの世代により異なる仕様のライブラリがあり, かつては SSE というライブラリが使われており, 最新のプロセッサでは AVX-512 が使われている. また, プロセシングコアを同時に利用してアプリケーションを書くためには, OpenMP [11] や POSIX スレッドライブラリ [12] などを用いて, 共有メモリシステムに対するマルチスレッドプログラミングをする必要がある. 一般に, POSIX スレッドライブラリを用いることで, 詳細なプログラミングが記述可能であるが, プログラミングが煩雑になる傾向があるが, OpenMP はプラグマを追加記述するだけなので, プログラミ

ングが簡便である. Intel の Knight's landing アーキテクチャのプロセッサは 72 コアをもつものもあり [13], マルチコアからメニーコアの時代へと変化しつつある. これらのアーキテクチャにおいてアプリケーションを作成するためには, 簡便なプログラミングが求められるので, OpenMP の重要性は高まっている.

一方, GPU は, CPU のプロセッサがアクセスするメモリとは異なる VRAM を用いて計算を行なうので, そのメモリ間の移動を含むプログラミングをすることになる. 現在の GPU の主要メーカーである NVIDIA 社の GPU を用いたプログラミングには, OpenACC [14] や CUDA [15] などを用いる必要がある. 前者は, マルチスレッドプログラミングにおける OpenMP のような位置づけであり, 簡便な方法で GPU 計算を実現するものであるが, GPU 計算の最適化は難しいので, CUDA を用いたプログラミングの必要性も低下しているわけではない. また, 複数のコンピュータを同時に利用して, さらに実行性能を向上させるクラスタコンピュータでは, 分散メモリ用のフレームワークである MPI [16] が利用されている. MPI であっても, CUDA であっても, 並列化する前のプログラムから, 大きくプログラムの改変を必要とするので, 初学者にとってはプログラミングが困難である.

2.2 初学者のための並列プログラミング学習

並列プログラミングに対する初学者にとっては, 逐次システムのプログラミングにはない, 並列システムの特徴により, プログラミングを学ぶ上での困難さが様々ある.

並列システムには分散メモリシステムと共有メモリシステムに分けられる. 共有メモリシステムは, 複数の計算主体 (プロセッサ, プロセシングコア, 演算器) がひとつのメモリを共有し, 時分割でアクセスするもので, 同時のアクセスは不可能であるものの, アドレス空間が共通なので, データの分割及び分散は不要である.

一方, 分散メモリシステムには, 複数ノード （コンピュータ）から構成されるマルチコンピュータシステムだけでなく, コンピュータ内にある GPU と CPU を使って計算を進めるような GPU 計算も含まれ, 後者では, CPU のメモリと GPU のメモリのアドレス空間が異なる. マルチコンピュータシステムではデータが分散にしているので, 元のプログラムにおけるデータを分割して, 分散しているメモリに配置し, 必要に応じて, メモリ間でデータを移動する必要があり, ユーザはそのためのプログラムの追加記述をしなければならないが. アドレス空間が複数, もしくは大量にある中で, データの分散状況を考えながらのプログラミングは初学者にとっては難易度の高いものになる.

また, 一般に, 並列プログラミングには 3 種類の手法があるとされている. 1 個目の方法は並列化コンパイラである. これは, コンパイラが既存の逐次プログラムから自動的に並列化できる部分を抽出し, ハードウェアにあった並列プログラムを生成するものであるが, 様々な試みがあったものの, ベクトル計算機を除き, 実用化レベルのものは存在しない. 2 個目の方法は, ディレクティブ方式で, 既存の逐次プログラムの中に, 並列化のためのヒントとなる指示句, すなわち, ディレクティブやプラグマを挿入するもので, 元のプログラムに徐々に追加することで並列化が可能であるので, 初学者には比較的容易な方法である. これに含まれるものは, OpenMP や OpenACC などがある, 利用者にとって, 比較的容易には見えるが, コンパイラは指示された通りに並列化を行うので, 利用者の理解が不十分であれば, 本来は並列化できない部分を間違って並列化する可能性があり, また, コンパイラがどのように並列化をしているかが見えにくいので, デバッグや最適化が難しくなる傾向がある. 3 個目の方法は, 指

定された API を用いてプログラムを改変する方法になり, これには CUDA, AVX, MPI 及び POSIX スレッドが含まれる. どのように並列化されるかは理解しやすいが, 一般には大幅な書き換えが必要となり, 一部分だけ並列化するということはできないので, 並列プログラムを完成するまでの時間的コストは高いものになる.

このように, 初学者にとっては, 対象とする並列システムの形態や, 並列化プログラミングの手法にとって, 理解すべき内容が多く, 難度の高いスキルが求められる.

2.2.1 共有メモリシステム

図 1 に示す共有メモリシステムとしては, メインメモリを共有するプロセシングコアを用いたプログラミングと, SIMD 演算器で演算する際にメインメモリを計算対象とするプログラミングの 2 パターンがある.

共有メモリシステムでの簡便な並列化では OpenMP の利用が一般的である. OpenMP は OpenACC と同様に, 既存のプログラムにプラグマを付加する形式であるので, インクリメンタルに並列化が行え, 初学者には取り組みやすい並列化手法である. さらに, データ並列的な並列化手法だけでなく, タスク並列的な並列手法も実現できるので, 共有メモリシステムにおいては事実上の標準技術といえる. しかし, プラグマの付加によりコンパイラが自動的にマルチスレッド化を行うので, 注意深くプログラミングを行わないと, 計算順が変更されたことによりデータ依存に基づく計算結果の不一致が生じる. 図 2 に OpenMP における計算順の変更例を示す.

```
int a[7]={0, 0, 0, 0, 0, 0, 0};
#pragma omp parallel for num_threads(2)
for (i=0; i<6; i++){
  x[i]=x[i+1]+1;
}
```

図 2: OpenMP における計算順の変更例

図 2 のプログラム例の場合, 並列化がされない場合は, x[0] から x[5] まで順に代入がされ, 代入文の右辺の x[i+1] はいずれも更新前の値が参照され, 代入結果はいずれも 1 となる. しかし, OpenMP のプラグマが有効になると, 2 個のスレッドが生成され, 1 つめのスレッドは x[0], x[1], x[2] の代入をこの順に実行し, 2 つめのスレッドは x[3], x[4], x[5] の代入をこの順に実行し, 2 個のスレッドが同時に開始されたとすると, 1 つめのスレッドが x[2] の代入をするときには x[3] の代入が実行された後の可能性が高く, x[2] は 1+1 となって 2 が代入されるかもしれない. スレッド間の実行タイミングはこのプログラムの情報だけでは決定できないが, ユーザはこのような計算間違いをしないようにプラグマの挿入しなくてならず, 必要があればプログラムの書き換えをしなくてはならない. このような留意点は初学者には理解しにくく, 一見容易に見える OpenMP プログラミングを難しくさせる要因になっている,

一方, より詳細な並列化を行う場合には, POSIX スレッドによるマルチスレッドが必要となる. 並列化の効果がよりよく出るのは, for ループのような部分であるが, POSIX スレッドでは, その部分を関数化し, その関数は, 1 個の引数しかとれないので, 必要な情報を構造体で表現する必要がある. 図 3 に示すように, 本来は, 配列 x を初期化する for ループだけであっても, その部分を関数として取り出し,

分散した計算の区間を示す情報を構造体を定義して表現し, その関数の引数に渡すことになる. ここには記載していないが, その構造体への初期設定の部分も書く必要があり, また, スレッドの停止を確認するための関数 (pthread_join 関数) を呼ぶ必要もあり, かなり煩雑なプログラムになる. このような書き換えは, 初学者にとってはかなりの負担である.

```
typedef struct parg{
  int bgn, end;
} parg;
parg p[10];
void sub (parg *p){
  int i;
  for (i=p->bgn; i<p->end; i++){
    x[i]=i*1.0;
  }
}
...
for (i=0; i< 10; i++){
  pthread_create(&th[i], NULL, &sub, &p[i]);
}
```

図 3: POSIX スレッドにおける関数化

最後に, AVX が計算の対象とするものは主に for ループの部分であるが, 複数の演算が同時に行なえ API を用いるので, for ループの回数をそれに合わせて減らす処理が必要となる. また, AVX レジスタに移動して計算することになるので, メモリと AVX レジスタとの間の移動は, アセンブラプログラミングのような書き方になる. いずれも, 初学者にとっては難しいものであると考えられる.

2.2.2 分散メモリシステム

マルチコンピュータシステムにおける並列プログラミング環境では MPI（Message Passing Interface）が標準的に用いられる [16]. MPI プログラミングの困難さは, 2 種類の分散をユーザが決定しないといけないことにある. すなわち, 単一のプログラムを複数のプロセスに分割し, さらにそれに伴ってデータをプロセス毎に分割することである. あるノード上のプロセスはそのノードのメモリのみが参照できるので, 異なるノード上のメモリは MPI の通信 API を用いてデータ交換が必要であり, 通信をいかに効率よくプログラミングするかが初学者には難しい点である. また, メッセージパッシングという仕組みもデッドロックを発生しやすいので注意が必要になる.

一方, GPGPU システムは GPU と CPU が物理的には別メモリ空間にあるので分散メモリシステムと考えられるが, GPU の複数のプロセシングコアはメモリを共有しているので共有メモリシステムとも考えられる. GPU をアクセラレータとして利用する計算環境を, GPGPU （General-Purpose computing on Graphics Processing Units）と呼ぶ. GPGPU では, CPU が用いるメモリから, GPU が利用する VRAM へ, データを移動する必要があり, そのデータ移動のコストを最少化する必要があるので分散メモリ的

であるが, GPU のコアは, VRAM を共有して計算を進めるので, 共有メモリ的な特徴もある. OpenACC は, GPGPU に対するプログラミング環境で, ベースとなる言語（C 言語など）にプラグマを追加する手法であり, 共有メモリマルチスレッディング技術の OpenMP のアプローチに似ているが, その性能向上は必ずしも十分でなく, 性能を引き出すには経験と慣れを要する. 特に, GPU のメモリと CPU のメモリ間のデータ移動のコストが, 性能に大きな影響を与えるので, そのデータ移動の最小化のために, 適切なプラグマを選択をすることは重要になる.

また, CUDA は NVIDIA の GPU 向けの技術であり, ベースとなる言語（C 言語など）に, 並列化やメモリ間転送に関する関数を追加する手法である. この手法は, GPU の持つ性能を引き出しやすいが, その手法を用いるために覚えるべき関数や考え方の修得が重要となる. CUDA の学習においては, 初学者のレベルと中級者のレベルにおいて, 異なる難しさがある. 初学者にとっては, 通常のシングルスレッドプログラミングとは異なるプログラミングパラダイムの理解と CUDA 独特な関数群の修得が難しい点である. 通常の C 言語の場合, プログラムの起動から終了までは一つの道筋, すなわちシングルスレッドであるが, CUDA における GPU 実行は, 明示的に指定しない限り, マルチスレッド実行であるので, 一つのプログラムがコア毎に異なる挙動をすることを理解する必要があり, スレッドの違いを区別するためのスレッド構成とスレッド ID 計算を理解する必要がある. また, CPU と GPU が論理的に異なるメモリを用いた計算を行うことも, 初学者にとって躓く点である. 一方, 中級者においては, 性能を向上するための最適化が難しい点になる. まず, メモリコアレシングは, メモリバンド幅を向上させるために ID 番号が連続するスレッドが連続するアドレスのメモリにアクセスする技術であるが, プログラム上では, アクセスするアドレスとスレッド ID 番号との関係を理解した上でプログラミングする必要があり, 複雑なプログラミングが必要となる. また, GPU は, SIMD（NVIDIA の用語では SIMT）で動作するので, 条件分岐によってスレッド間で異なる分岐をする（ブランチダイバージェンスと言う）と実行時間の増加を生じる可能性があるので, IF 文などの条件分岐を生じる可能性のある文を削除するか, もしくはそれらを適切に配置することによって, ブランチダイバージェンスを削減する工夫が必要である.

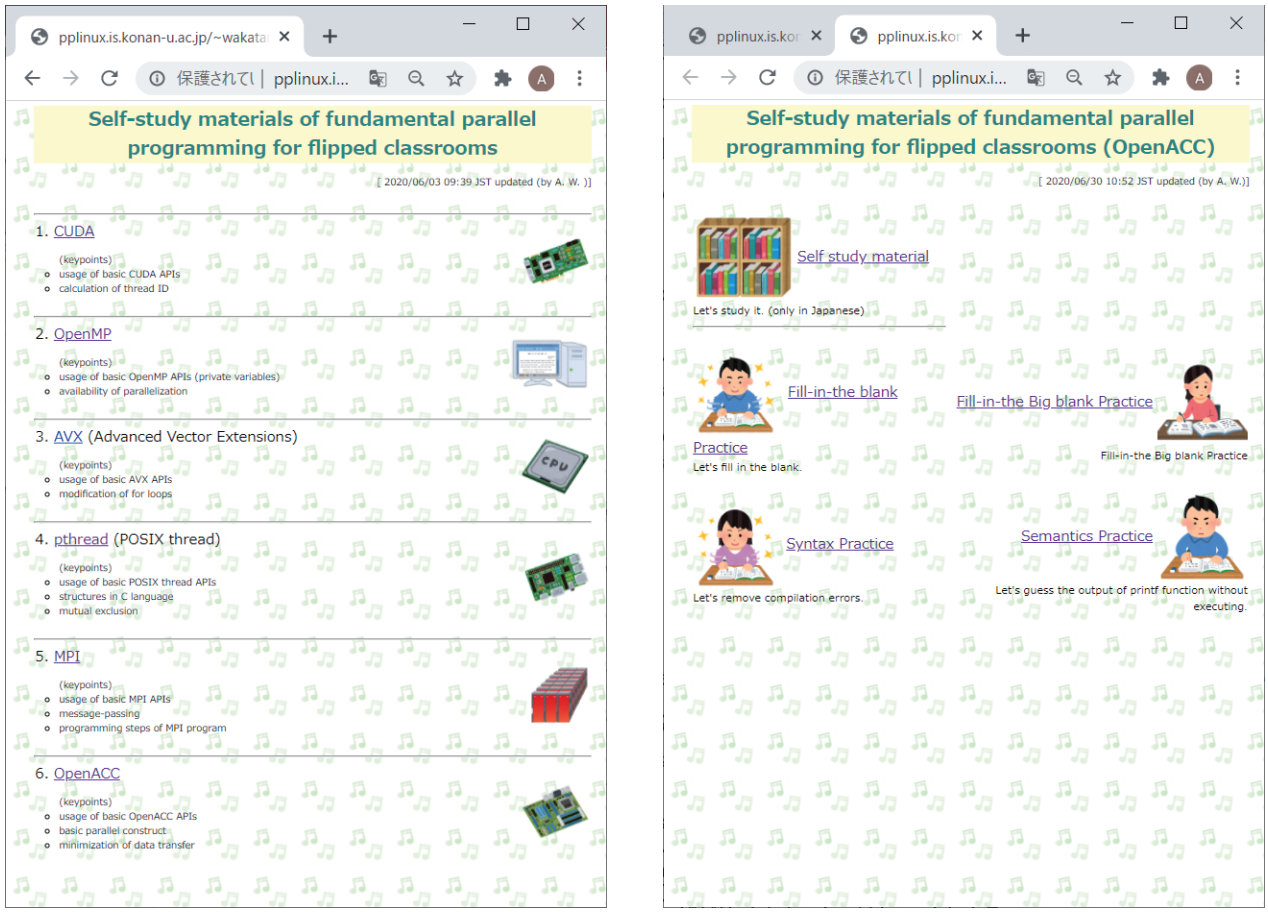
3 Web アプリケーションスイート

前章で述べたような初学者にとって難しい点を考慮し, 自宅での準備学習や, 自習用として, 並列プログラミングのための Web アプリケーションスイートを構築した. このアプリケーションスイートには, OpenMP, POSIX スレッド, AVX, CUDA, OpenACC, MPI の 6 種類の並列プログラミングに対応している. 図 4 に, トップページ及び OpenACC に対するサブページの様子を示す. なお, OpenACC 以外の 5 種類の並列プログラミングに対しても, 同じレイアウトのサブページが作成されている.

サブページには 5 個のリンクがあり, 1) 疑似動画教材 (self study material), 2) 単語レベルの穴埋め問題 (fill-in-the-blank), 3) 行レベルの穴埋め問題 (fill-in-the-big-blank), 4) プログラムデバッグ問題 (syntax practice), 5) 出力推定問題 (semantics practice) がリンクされている. 概ねこの順番に学習を進め, まず, 疑似動画教材で, その並列プログラミングの概要を学習したのちに, 4 個のプログラミング演習ツールを使って, 学習した内容の理解を深め, 知識を固定化していく. なお, プログラミング演習ツールは, 難易度の順に配置してある.

現在 (2020 年 8 月), 6 種の並列プログラミングに対するサブページが完成し, 一部の実証実験を開始

し, OpenACC プログラミングを除く 5 種の並列プログラミングのページは, 対面講義自粛期間中の自宅学習用の Web 教材として利用した.



(a) トップページ　　(b) OpenACC サブページ

図 4: Web アプリケーションスイート

4 MPI 及び OpenACC プログラミング学習ツール

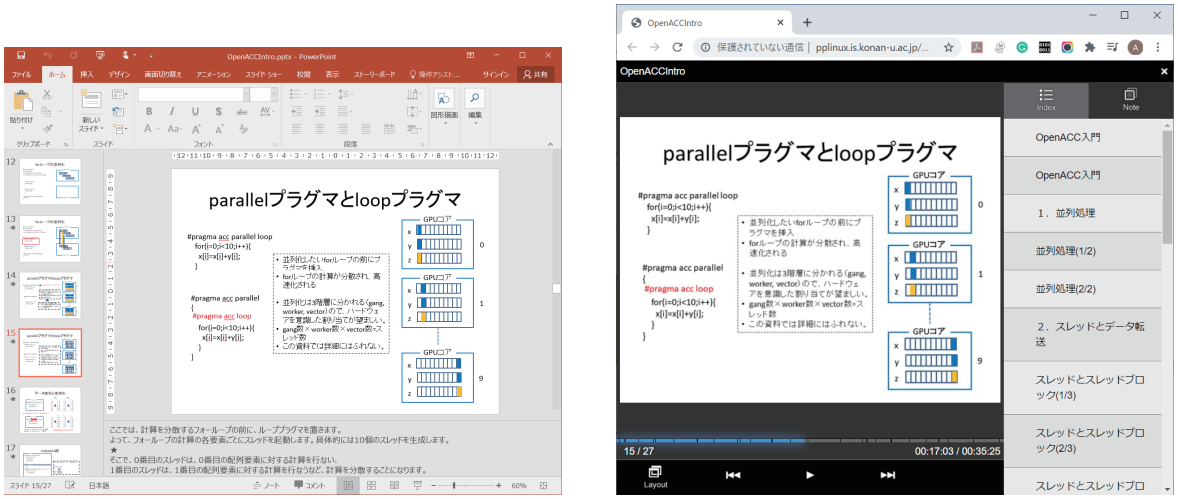
今回の Web アプリケーションスイートの中で, MPI 及び OpenACC プログラミングに対する疑似動画教材と, 各種プログラミング演習のページを, 以下のポリシーで作成する. 1) 作成コストを大幅に削減した事前学習教材を生成すること, および 2) ソースコードを擬似的に書くような演習問題を自動的に生成し, その採点も自動的に行えること, が実現できる方式を用いる. プログラミング演習では, 初学者に対する並列プログラミング教育における例題を, 1) 文法理解のためのプログラムデバッグ問題, 2) プログラム意味理解のための出力推定問題, 3) 上記の要素を総合的に含んだ穴埋め問題, の 3 つの演習に集約し, 簡単で採点が容易なレベルの例題に注目し, その例題をテンプレートを用いて自動生成する Web アプリケーションを PHP 言語を用いて開発することを試みている. これらの演習問題は, 正

解から生成されているので, 自動採点が容易にできる. なお, 穴埋め問題には, 単語レベルの穴埋め問題と, 行レベルの穴埋め問題を用意し, 難易度に合わせて使い分けするようにしている.

4.1 疑似動画教材

疑似動画教材は PowerPoint を用いて作成する. 内容は, それぞれの並列プログラミングの文法や API の概要と実行時の典型的な実行に関するものである, 例えば, OpenACC であれば, 使用されるプラグマと, ホストとデバイス間のデータ転送の必要性と, その最小化を説明し, MPI であれば, データ分散方式と, データ転送のための API の仕組みと使い方を説明することになる.

疑似動画とするために, PowerPoint 資料はアニメーション機能を使い, また, 音声と同期するようにして. ユーザに分かりやすいものとする. 音声は, STORM Maker（現在は, STORM Xe） [17] の音声合成機能を用いる. すなわち, PowerPoint のノート部分に書いた文章を音声合成し, PowerPoint の動画再生と同期して表示する機能である. 図 5 に, OpenACC におけるプラグマの説明に対する, PowerPoint の例とコンテンツの例を示す.



(a) PowerPoint　　(b) 表示

図 5: 疑似動画教材の例 (OpenACC)

図 5 (a) の PowerPoint のノートの中で “★” のある部分は, 表示におけるアニメーションの切り替えタイミングを示している. これによりナレーションの開始するポイントと, アニメーション表示の同期が実現でき, 疑似的な動画教材として提供することができる.

なお, 音声合成は自動でされるが, 読み方が異なると手動で文章を修正することが必要となる. 例えば, 「あたい（値）」と発声させたいケースにおいて「値」と書くと, 「ね」と発音されるので, 漢字の「値」ではなく, ひらがなで「あたい」と記述する. これ以外にも, 様々な微調整が必要になるが, 一般的には, 文章入力のみなので, 自分で録音するのに比べて, 編集のコストは大幅に削減できる.

4.2 プログラミング演習教材

並列プログラミングの教育においても, 他のプログラミング言語の場合と同様に, 知識の獲得だけではなく, 実際にソースコードを作成して学ぶことが重要である. しかし, 対面講義自粛期間の自宅学習では, 自宅でひとりで学習する際にソースコード作成の演習およびその採点を行うことは困難である. そこで, ソースコード作成に近い疑似的な演習を行うことを考える. まず, 動作がわかっている正解のプログラムを生成し, そのプログラムに対して, 以下に述べるのような, 様々な演習問題を作成していく. 正解のプログラムは少数であれば演習問題の総数が限られるので, 実行時に多数のプログラムに生成するように, テンプレートと置き換え手法の組み合わせで実現する.

対象とする演習内容について考える. プログラミング教育の基本は文法の習得, 特に API やプラグマの習得である. そこで, API やプラグマ理解の補助教材として, API に関するバグを含むプログラムを学習者に提示し, それを修正することにより正しいプログラムにする “プログラムデバッグ問題” を用意する. また, 並列処理向けプログラムの作成においては, マルチスレッドやマルチプロセスの概念を理解し, どの計算要素がどの順番で実行されるのか, データ依存が保存されるのか否かをソースコードを読むことから理解することが必要となる. その能力を涵養するために, “出力推定問題” では, 意味理解の補助教材として, プログラムの挙動を推定させる. さらに, 穴埋め問題においては, 上記の文法的理解および意味論的理解の双方の能力を高めるために, プログラムの一部を書き込むという穴埋め問題で実現していく. 穴埋め問題は, 前者の 2 つの演習の前に, 予備的に, かつ, 十分行うことが重要となる. 穴埋め問題には, 単語レベルのものと行レベルのものを用意し, 学習者の理解度にあわせて, 難易度を選べるようにする.

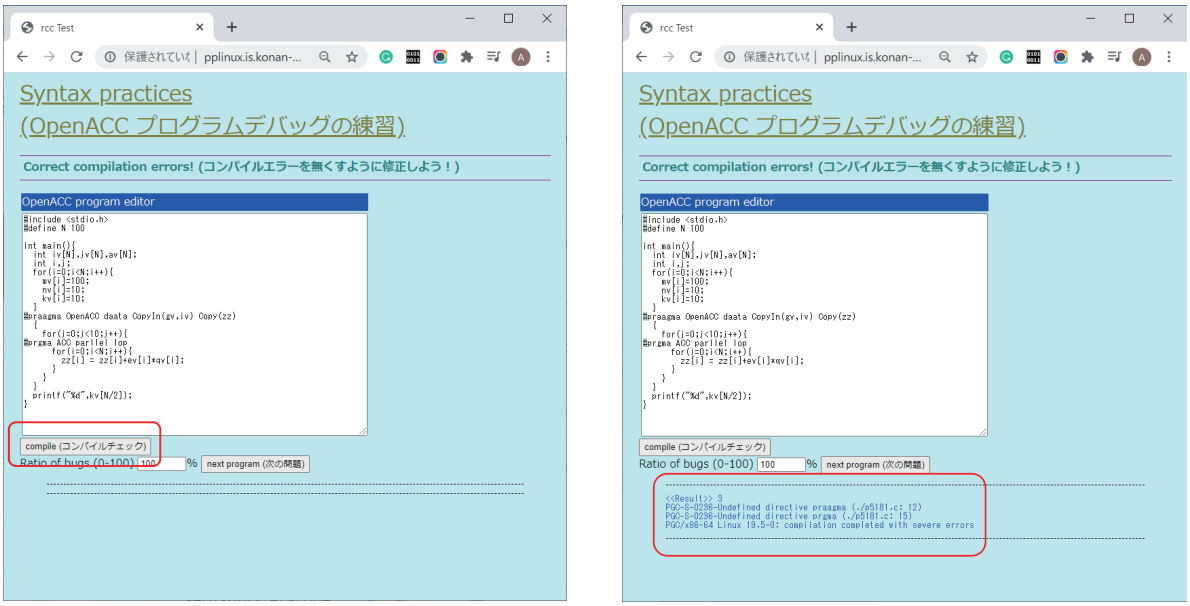
実装に関しては, Web アプリケーションとして実現するために, PHP を用いて作成している. ユーザに大量の演習問題を提供するために, 少数（10 個程度）のテンプレートをもとに, 変数名, 配列名および一部の演算式をランダムに選択するようにして問題生成を行う. また, 採点に関しては, 必要であれば実行時にサーバ内で解答を生成し, ユーザの解答との比較をするようにしている. 実装の詳細は文献 [18] などに記載している. なお, 以下の部分で図として表示しているのは OpenACC プログラミングでの演習教材であるが, MPI プログラミングを含む他のものについての演習教材も同様の表示になっている.

4.2.1 プログラムデバッグ問題

プログラムデバッグ問題のプログラミング演習教材は, API や文法の理解の定着を図るためにバグを含むプログラムのデバッグを繰り返し行い, 主に API 文法理解の間違いを修正することを目的とする. 図 6 に本補助教材の実行画面を示す. OpenACC においては, プラグマを書くことが主なプログラミング内容であるので, プラグマのキーワードについての正しい理解を促すような問題の内容にする. また, MPI においては, 各種関数の名前などの理解を確認するような問題内容になっている.

プログラムの内容としては, OpenACC プログラミングの場合は, 単純な for ループを parallel プラグマと loop プラグマで並列化するだけなく, data プラグマで GPU メモリと CPU メモリ間のデータ移動をさせるために copy 節, copyin 節及び copyout 節を使うようにしており, 関数呼び出しの際にデータ移動を抑制するための present 節や, 総和計算のようなリダクション計算を行なうための reduction 節を利用したプログラムも含んでいる. また, MPI プログラミングの場合は, MPI_send 関数と MPI_recv

関数を組み合わせた単純なメッセージパッシングだけでなく, プロセス ID (rank) の計算を行なって分散計算をするものを含んでいる.



(a) 初期画面　　(b) コンパイルチェック

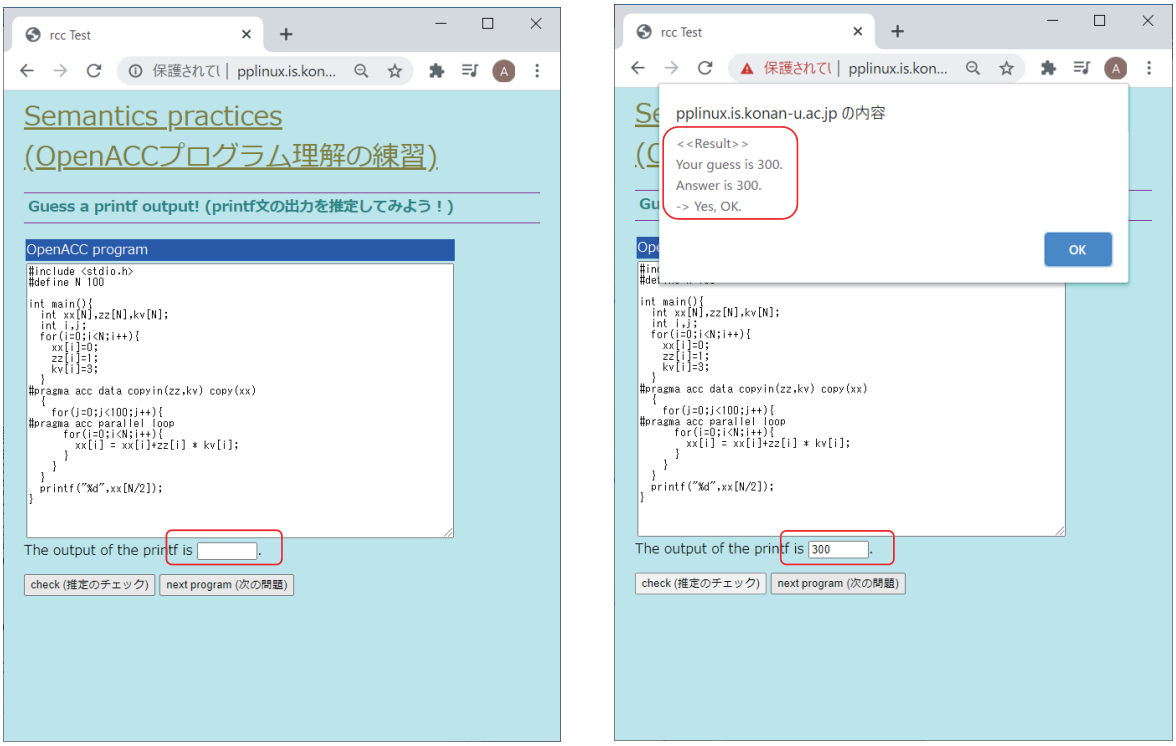
図 6: プログラムデバッグ問題 (OpenACC)

本補助教材では, まず, 中央の枠内にバグを含むプログラム例が表示される (図 6 (a)). このまま “compile check” ボタンをクリックすると, コンパイルエラーが表示される (図 6 (b)). ここでのエラーは, プラグマのキーワードのスペル間違いが指摘されている. 学習者は, 理解している API の知識とエラーメッセージを参照してプログラムを修正し, エラーがなくなるまでチェックを続け, 最終的にバグが無くなると文法的に正しいプログラムになる. さらに学習を続ける場合は, “next program” ボタンをクリックすると, 新しいバグのあるプログラムが表示される. なお, 新しい問題を提示する際に, バグ率 (ratio of bugs) を変えることにより, バグの含有率を変えることができる. バグ率を 0 にするとバグの無いプログラムが提示され, 100 にすると, あらかじめ想定していたバグのすべては含まれるプログラムとなる. しかし, バグ率が高過ぎるプログラムであれば, 本来の正しいプログラムが見えづらくなるので, デフォルトは 8 から 30 に設定している.

ユーザがデバッグする際には, 必要に応じてコンパイラのエラーメッセージを利用することが一般的である. API を含む文法の理解はエラーメッセージに頼らないようにすべきという考え方もあるが, 実践的場面においては, プログラミングはコンパイラを利用することが普通である. したがって, エラーメッセージを参考に, 正しい並列プログラムへ変更することにより, 理解し間違えていた API を正しく理解できるようになる. また, バグ率を変更できるので, より深い理解をしたい場合は, バグ率をあげて教育効果の効率を向上することも可能である. 以上のことより, 教材を読むだけで文法を理解するのではなく, 一定の実践的な例題プログラミングを課題として行うことができ, 並列プログラミングの能力の定着が図れると期待できる.

4.2.2 出力推定問題

出力推定問題のプログラミング演習教材は, OpenACC プログラミングにおいて, 計算の結果がどのようになるかを確認し, 特にデータ移動に関するプラグマの結果が, 計算結果にどう影響するかを推定するかを理解することを目的とする. また, MPI では, 分散メモリ間でのデータ移動と計算結果の関係を考えさせるような問題としている. 図 7 に, OpenACC プログラミングに対する本補助教材の実行画面を示す.



(a) 初期画面 (b) 出力推定

図 7: 出力推定問題 (OpenACC)

本補助教材では, まず, 中央の枠内にプログラムが表示される (図 7 (a)). 学習者はこのプログラムを読み, その意味を理解した上でプログラムの後半にある printf 関数の出力を推定する. 推定した値を解答欄に記入後, “guess check” ボタンをクリックすると, 実行結果との比較が表示される (図 7 (b)). ここでの計算は, 1 と 3 に初期化された配列 zz と kv が copyin で GPU メモリに移動され, GPU において, それらの乗算と 0 の加算結果を配列 xx に代入し, それを 100 回繰り返ししている. さらに, 配列 xx は copy で CPU メモリに移動されているので, printf 関数の表示は 300 となり, それを回答すれば正解になる. 表示されるプログラムには, OpenACC プログラミングであれば, プラグマによる CPU メモリと GP メモリ間のデータ移動を含むものであり, MPI プログラミングであれば, 分散されたメモリ間のデータ転送とプロセス ID を含んだもので, 暗算での解答が難しいものがあるが, 並列プログラムの意味を理解する能力の涵養を図ることができるものになっている.

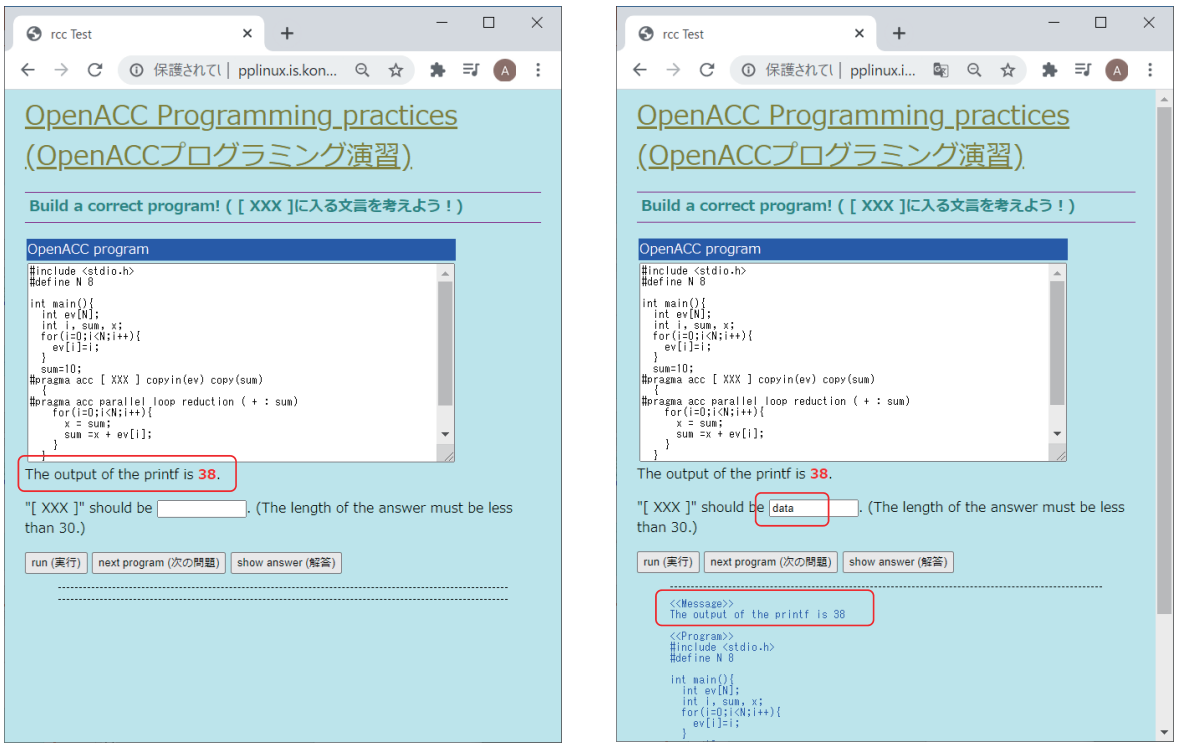
プログラムの内容としては, プログラムデバッグ問題と同様に, OpenACC プログラミングの場合は, data プラグマ, parallel プラグマ及び loop プラグマを含むもので, MPI プログラミングの場合は, メッ

セージパッシングとプロセス ID (rank) の計算を組み合わせたものになっている.

なお, 問題のパラメータ設定によれば, ゼロ割り算や未割当てのメモリへのアクセスエラーなどを発生させる可能性があるので, 実行時のエラーが生じないように留意する必要がある.

4.2.3 穴埋め問題

プログラムデバッグ問題や出力推定問題を最初に行うことは, 初学者にとっては困難な場合がある. そこで, その両者の能力を予備的に涵養するために, 穴埋め形式の問題も演習として提供する. 図 8 に本補助教材の実行画面を示す.



(a) 初期画面　　(b) 結果確認

図 8: 穴埋め問題 (OpenACC)

なお, 穴埋め問題には, 単語レベルの穴埋めと, 行単位の穴埋めの 2 種類を用意している. 前者の方が簡単であるので, 学習においては, 前者での演習が終わったのちに, 後者に取り組むという順序で行なう.

図 8 (a) に示すように, プログラムの出力があたえられており, その出力になるように XXX に書くべき文言をユーザが答える形式の穴埋め問題である. ユーザはプログラムの意味を考えて XXX を考えるだけでなく, 文法上正しい記述方法を考えることになる. また, 穴埋めは 1 箇所に限定しているので, ユーザには他のソースコードの部分を読む機会が与えられている. よって, ユーザはより多くのソースコードを読むことになるので, 早く, 教育対象となっている並列プログラミングのスタイルに馴れることができる. ここでの問題は, プラグマのキーワードを答えるもので, data と入れれば正解になる.

プログラムの内容としては, プログラムデバッグ問題と同様に, OpenACC プログラミングの場合は, data プラグマ, parallel プラグマ及び loop プラグマを含むもので, MPI プログラミングの場合は, メッセージパッシングのための送信と受信の関数の記述方法に関する内容と, プロセス ID (rank) の計算を組み合わせたものになっている.

なお, 実装においては, 穴埋めのところに任意の文字列を書くことができるので, 1) 無限ループのプログラムになった場合に強制終了できるような仕組みの実装, および 2) Web アプリケーションとしてセキュリティ上の問題が生じないような工夫が重要となる.

5 MPI プログラミングの教育に対する予備評価

これまでに, CUDA プログラミング [19] や OpenMP プログラミング [20] に関する教材の作成を行い, 予備的な評価を行ってきており, Web アプリケーションを用いたプログラミング演習の実施が, 並列プログラミング対して, 教育的効果を上げていることが示されている. ここでは, MPI プログラミングに対する疑似的な反転授業の効果を測定した結果を紹介する. MPI は, 複数コンピュータので構成される分散メモリシステムに対する並列化プログラミング環境であるが, 本システムでは, サーバー内で OpenMPI の環境を動作させ, 4 個プロセスを用いて疑似的に分散実行を行っている.

今回の予備評価では, 穴埋め問題のプログラミング演習のみを行い, それにより MPI プログラミングの理解がどれくらい進んだかを調べた. 実験の構成は以下の通りである. 1) 約 30 分の疑似動画教材で MPI プログラミングについての学習を行い, 2) 事前テスト（5 分）で理解度を測定し, 3) 穴埋め問題のプログラミング演習を 20 分間行ない. 4) 事後テスト（5 分）で理解度を測定し, 比較する.

事前テストおよび事後テストも, 文法的知識を問うための API に関する質問と, 意味的理解を問うための, データ転送関数のプロセス ID (rank) の計算の結果に関する質問から構成され, 難易度はほぼ同程度とした. API に関する質問には, API 名のスペルミスや, 関数やヘッダファイルの名前間違いが含まれ, 文法的事項と意味的理解の事項の割合も同程度とした. 被験者は, C 言語については一通り学習しているが, 事前に MPI プログラミング経験はない学生 6 名とした. 表 1 に実験結果を示す.

表 1: プログラミング演習の効果（MPI）

	pre	post
1	100%	100%
2	50%	100%
3	50%	80%
4	100%	80%
5	80%	100%
6	90%	80%
total	80%	90%

6 人中 4 人は, 事後テストの結果が事前テストより上回る, もしくは同じであったので, 穴埋め問題のプログラミング演習をすることで, MPI プログラミングの理解を進めるための, 一定の効果は認められると考えられる. なお, テスト結果において, 間違いの多くはプロセス ID の計算に関する部分で

あり, 予想されることではあるが, この部分が初学者にとって難しいことが確認できた. 実験では, 限られた時間で, かつ, 穴埋め問題のみを使っており, 被験者数も限定的であり, より本格的な実証実験が望まれる.

また, 2020 年の前期においては, 対面講義が自粛期間であり, 本 Web アプリケーションスイートを用いて講義を行った. 具体的には, 5 週にわたって, OpenMP, POSIX スレッド, CUDA, MPI, AVX のプログラミングのオンライン講義として使用した. 実機でのプログラミング演習を十分に行える環境が提供できない条件では, このようなシステムは有効に活用できることが確認できた.

今後は, OpenACC を含む, 6 個の並列プログラミングの教育に対して, 実証的な検証を進め, 本アプリケーションスイートの有効性を確かめるとともに, 実用的な活用ができるように改良を続けていきたい.

6 おわりに

本稿では, 並列プログラミングの主要技術である, OpenMP, POSIX スレッド, AVX, MPI, CUDA, OpenACC のプログラミング教育のための学習教材を, Web アプリケーションスイートとして試作するためのポリシーと, 完成した試作版の現状及びその予備評価について述べた. また, MPI プログラミングの教育に対する予備評価の結果より, プログラミング演習の Web ツールは理解度の向上に寄与することが確認できた.

本 Web アプリケーションスイートは, 初学者の自習用教材として用いるだけでなく, 反転授業における事前学習にも用いることができ, さらには, 新感染症対策の対面授業自粛期間における, オンライン講義の教材としても利用することができる.

今後は, 教育効果を高められるようなプログラミング演習の改良を行うとともに, MPI や OpenACC などの学習効果について, 実証実験を進めることで本システムの有効性を確認し, ソフトウェア教育へ貢献していきたい.

謝辞

本研究の一部は JSPS 科学研究費 (基盤研究 (C) 18K02920 (2018-2020), 基盤研究 (C) 19K03018 (2019-2021)) 及び私立大学等経常費補助金特別補助「大学間連携等による共同研究」による.

参考文献

[1] J. バーグマン, A. サムズ著, 山内祐平訳, 反転授業. オデッセイコミュニケーションズ, 2014.

[2] J. Wesley Baker, “The classroom flip: Using web course management tools to become the guide by the side,” in *Proc. 11th International Conference on College Teaching and Learning*, pp. 9-17, 2000.

[3] http://images.nvidia.com/content/technologies/deep-learning/pdf/NVIDIA-Tesla-V100-JPN.pdf (as of 2020/8/21).

[4] https://chainer.org/ (as of 2020/8/21).

[5] M. Maher, H. Lipford and V. Singh, "Flipped classroom strategies using online videos," http://maryloumaher.net/Pubs/2013pdf/Flipped-Strategies-CEI-Report.pdf, 2013.

[6] D. Radošević, T. Orehovački and Z. Stapić, "Automatic on-line generation of student's exercises in teaching programming," in *Proc. Central European Conference on Information and Intelligent Systems*, CD-ROM, 7 pages, 2010.

[7] S. Gupta and S. Dubey, "Automatic assessment of programming assignment," *Computer Science & Engineering: An International Journal*, vol. 2, no. 1, pp. 315-322, 2012.

[8] R. Queirós, J. Leal, S. Gupta and S. Dubey, "Programming exercises evaluation systems: An interoperability survey," in *Proc. 4th International Conference on Computer Supported Education*, pp. 83-94, 2012.

[9] A. Wakatani, http://pplinux2.is.konan-u.ac.jp/rccE/ppindex.php (as of 2020/8/21).

[10] https://www.intel.com/content/www/us/en/architecture-and-technology/avx-512-overview.html (as of 2020/8/21).

[11] https://www.openmp.org/ (as of 2020/8/21).

[12] https://computing.llnl.gov/tutorials/pthreads/ (as of 2020/8/21).

[13] https://ark.intel.com/products/codename/48999/Knights-Landing (as of 2020/8/21).

[14] http://www.openacc.org/ (as of 2020/8/21).

[15] https://developer.nvidia.com/cuda-zone (as of 2020/8/21).

[16] https://www.mpi-forum.org/ (as of 2020/8/21).

[17] https://suite.logosware.com/storm-maker/ (as of 2020/8/21, in Japanese).

[18] A. Wakatani and T. Maeda, "Evaluation of software education using auto-generated exercises," in *Proc. the 1st International Workshop on Emerging Technologies using Computer Sciences*, USB, 4 pages, 2016.

[19] A. Wakatani and T. Maeda, "Web-based applications for learning GPU programming," in *Proc. 8th International Conference on Information Intelligence Systems Application*, CD-ROM, 5 pages, 2017.

[20] A. Wakatani and T. Maeda, "Web applications for education of parallel programming," in *Proc. Society for Information Technology & Teacher Education 2018*, CD-ROM, 6 pages, 2018.